Random data entropy in OS installations

Vratislav Podzimek
Faculty of informatics,
Masaryk university,
Brno

25.05.2013

Introduction and basic terms

Many areas of IT security rely on having enough and high-quality random data [1]. Typical are cryptographic operations – initialization vectors for various symmetric encryption/decryption algorithms, prime numbers generation for asymmetric algorithms, padding used by both symmetric and asymmetric cryptography and also e.g. hash functions, salting used when storing hashed passwords, random nonces in various protocols, etc. But there are many other not so obvious cases like memory placement of code [2] and data when running a binary or assigning ports where random data play an important role in overall security of a system. Ideally, it would be best to use truly random data generated by a *Random Bit Generator* (RBG), but unless there is a special device used (e.g. Quantum RBG [3]), it is practically impossible to get enough truly random data that is needed by operations done on a typical system, be it a workstation where user visits web sites over the HTTPS protocol and communicates via some instant messaging system using SSL, or a server establishing thousands such connections per second. Thus so-called *Pseudorandom Bit Generators* (PRGB) are used that for l bits of truly random data (a *seed*) provide k bits of pseudorandom data where $k \gg l$.

Pseudorandom bit generators differ in algorithms they use (often based on some cryptografic operations done on the seed) and in the implementations of such algorithms (software or hardware). These differences result in different quality of the generated data and also the speed the data is generated with (in bytes per second). But what all PRBGs have in common is that the quality of the generated pseudorandom data depends on the quality and amount of the seed data. The better the quality of the seed (random) data, the better the quality of the pseudorandom data and a similar rule applies to the quantity of the random data and quality of the pseudorandom data. So even though PRBGs (partially[1]) solve the problem with the availability (quantity), to provide high-quality pseudorandom data the availability of truly random data is still crucial.

Typically the systems (or more precisely kernels of systems) gather random data from a "noise" registered by various devices. In case of e.g. wi-fi or ethernet adapters it may be a real noise (as known in physics), but in case of some other devices such as keyboard and mouse, it is the side-channel information like timing of keystrokes and movements that is used. All these resources are accumulated and combined together to provide random data. However, there are some machines and devices, that are limited by their hardware. E.g. the mobile phones are often left in the idle mode where such sources of random data are not in action (of course the antenna and transmission controller do something)[2] or there are servers with no mouse, no keyboard and no wi-fi adapter that might run into problems with random data availability.

Because gathered random data is such a rare and precious resource[3], what kernels of operating systems (OS) usually do is that they store the so-called *random data pool* on reboot to have some random data to start with during boot where the situation would be otherwise critical due to very short time available for gathering random data. However, there is at least one case when the system boots and the pool has to be empty – the OS installation. One may ask why there cannot be a pool stored on the installation DVD or network resource, the installation is performed from. The answer is that the DVD is produced once and then used for many installations and thus all the installations would have started with the same pool which would lead into similarities in their "random" data. Providing a different pool on each DVD or amending it to the resources downloaded via network seems to be a practically impossible task and would be limited by the resources of the machine producing such pools. So the installation practically has to start with an empty pool of random data. Another question is if random data is actually needed in the installation process. And the answer here is that yes, it is because e.g. many keys may be generated during installation – disk encryption master key, SSH keys, keys for the SSL certificates used by the

[1]the `/dev/urandom` file on a GNU/Linux machine I use to write this report generates cca. 1.7 MB of pseudorandom data per second which may be not enough in many cases

[2]In case of the mobile phones (but hardly the new shiny smart phones) a problem with computational performance and energy consumption step into play when generating pseudorandom data [1].

[3]the `/dev/random` file provides cca. 10 B/s on my machine

web server and so on. And for example as the most probable issue behind the recent findings that many servers have the same RSA keys [4], a low quality of random data during their installations has been identified.

The previous paragraphs multiple times refer to quality of (pseudo)random data. Intuitively this means how much the data are unpredictable and indistinguishable from truly random data (produced e.g. by the Quantum RBG the correctness of which is based on physical laws). To measure quality of random data, two methods are usually used – computing *entropy* or performing statistical tests. The statistical tests of random data is a wide area that is out of the scope of this work. The *entropy* is computed with the following formula [1]

$$E = -\sum p_i log_2 p_i$$

where p_i is the probability of appearance of the i-th element (from a set of possible elements), and gives the amount of information contained in the data and also predictability of the data. If some element appears more often than others in some sequence (which means lower entropy when compared to the even distribution), it more likely to appear as the next item. Although, as it can be easily seen, it is trivial to come up with non-random data that has maximal entropy, higher entropy of the seed is expected to result in better quality of the pseudorandom data generated from it.

2 Gathering data

As has been described in the previous section, a potential problem with the quality of (pseudo)random data may appear in the installation process. But OS installation is a specific environment and in case of closed-source operating systems, it is practically impossible to inject any additional code or tool next to the OS installer. Fortunately, there are open-source operating systems that provide quite easy way to achieve that. For example the installer of Fedora and RHEL GNU/Linux distributions[4], the *Anaconda installer* [5], provides a way to use a special file, so-called *updates.img* [6] that the installer unpacks and places to the environment. As the name suggests, it is usually used to test updates of the installer itself, but it can contain anything. Thus, by using such special file, it is possible to inject some code that gathers data for further analysis.

The other question is what data should be gathered to allow making some conclusions about the random data quality in the installation process. One of the key features for the pseudorandom data (the data actually used by the

[4]and a number of their derivatives

tools) quality is the overall entropy of the random data pool managed by the kernel. Being quite important metric, the Linux kernel provides APIs to get this value. One is the special file `/proc/sys/kernel/random/entropy_avail` that contains the current value of overall entropy in bits as the only line. The other way is the `RNDGETENTCNT` `ioctl` call [7] on an opened file descriptor for the `/dev/random` file that returns the same value, but in contrast to the first method there is no need to open and close the file every time.

When writing a simple code using the latter API to get current available entropy, we have found out that the value fluctuates over time. Thus it was decided that not only the level of entropy at the beginning of the installation should be gathered, but the whole installation procedure should be analyzed from this point of view. To gather such data a simple profiler (the **entropy-profiler** [8]), that just periodically (by default every second) gets info about the current available entropy from the kernel and writes it out, has been created. And since it could produce quite a lot of data if run for a long time, the output is compressed with the `zlib` library[5]. Once started with a single argument, the output file path, it runs until it gets the `SIGTERM` or `SIGINT` signal which means it should close the output file and terminate. In the beginning of the output there is a line containing a timestamp and info about the sampling interval. This line is followed by entropy values each on a single line with every fifth line having a timestamp. To allow starting the profiler even before the Anaconda installer starts (because that's the first time where some changes to the target system may potentially happen) a **systemd** service has been created. Once started it simply runs the profiler with the `/tmp/entropy_profile.txt.gz` file path and when stopped, it sends the profiler the `SIGTERM` signal. This service was then hooked up to the Anaconda's service in a way that it is started before the installer and an invocation of the **systemctl** utility to stop the profiler has been added to the installer's code. The profiler, it's service and updated files of the Anaconda installer were then put into an updates.img file passed to the installer. At the end of the installation the output file was copied from the installation environment and another run was started.

Because it was expected that some code would have to be added to the Anaconda installer (written in Python) to make sure there is enough entropy, a Python module (written in C) [9] has been created to allow that. It uses the same API as the profiler, but wraps it with the code creating Python object for the result (or raising an exception in case of an error). This module was used for some experimenting, but in the end it was replaced by a simpler code using the special file under `/proc` because when used only once in a second,

[5]used e.g. by **gzip** tool

there were no performance issues with such approach that, in contrast to the C code, doesn't require compilation and a global variable holding the opened file descriptor.

The entropy level is one thing, but what's important is the quality of the (pseudo)random data the kernel gives to the tools that request them. For this reason, some samples of data provided at the very beginning of the installation process were gathered and analyzed. The standard **dd** utility was used to "copy" 32 MB from the `/dev/urandom` special file to a file under `/tmp` directory that was then copied to a different system during the installation process.

3 Data evaluation

Data was gathered from four machines, three virtual machines (two using hardware virtualization and one using emulated virtualization) and a laptop, and were then analyzed in multiple ways. The first way was simply a manual analysis of the numbers returned by the entropy profiler. Several results were observed that way, but for analysis of evolution of some values, a graph is better than a list of numbers and thus graphs were plotted for all entropy profiles. The examples follow and many interesting results can be easily observed from them.

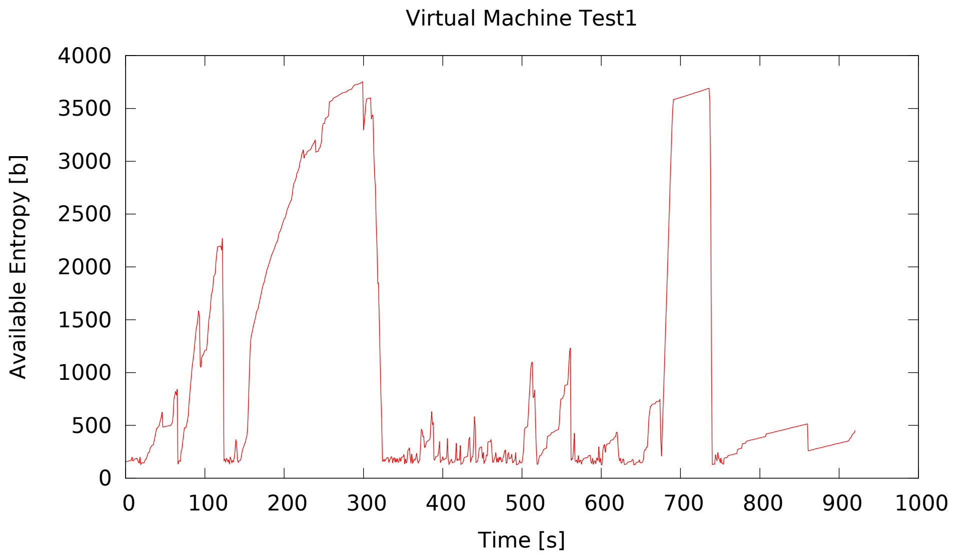

Figure 1: Entropy profile from the virtual machine Test1

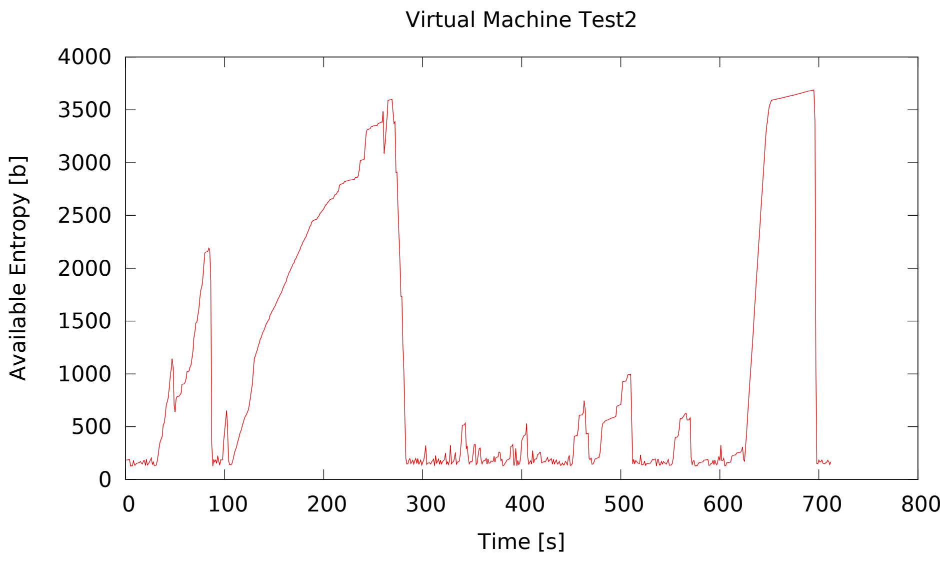

Figure 2: Entropy profile from the virtual machine Test2

What is obvious at the first sight is that all graphs follow some pattern. This is caused by the actions which are performed in particular phases of the installation process. For example storage setup and configuration (creating partitions and filesystems) is implemented via calling many tools used for such operations (**mkfs**, **parted**, **vgcreate**, ...) which means that the entropy level falls down quickly due to kernel's stack protection code. On the other hand, package installation[6] doesn't fork any additional process (or at most one) and puts heavy load on the disks, that are one of the sources of truly random data (seek times, etc.), which means that the entropy level rapidly raises. The smaller peaks are probably results of user interaction. On the other hand the pits are probably caused by the installer forking many processes.

Interesting outcome is also the fact that the virtual machines have more entropy than the laptop despite the common expectation that such machines must have problems with random data quality due to the lack of real hardware. However, such common expectation is wrong for many reasons. First of all, designers of virtualization technologies are aware of such possible problem and thus virtual machines (guests) are typically "equipped" with a special device [10] connected to the random data pool of the host. Moreover, recent hardware virtualization technologies allow direct usage of the CPU as well as of the storage, and some other, devices [11]. Thus the situation of a virtual

[6]mainly downloading packages, the final verification and cleaning up temporary data

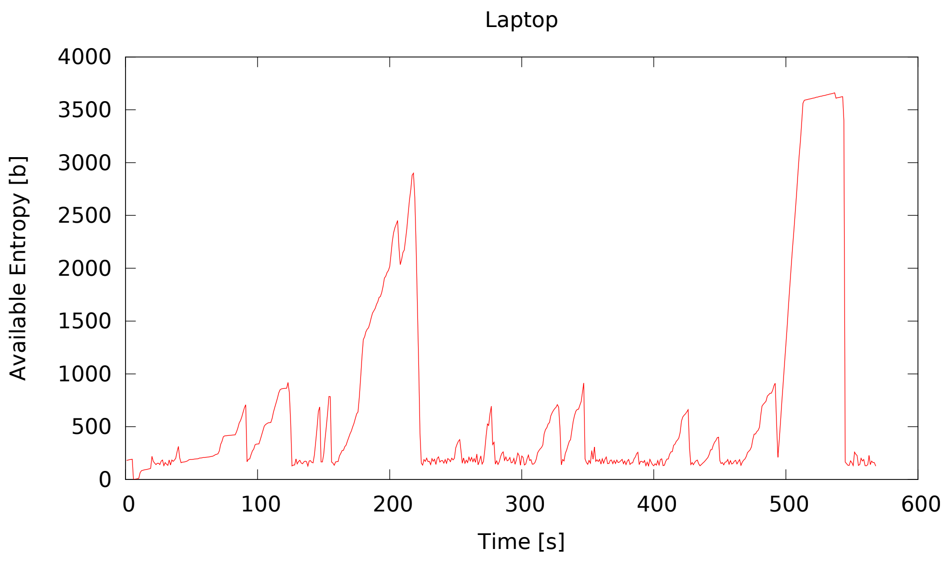

Figure 3: Entropy profile from the laptop

machine is actually better than the situation of a physical machine and the gathered data only confirms that as the laptop starts with almost critical values where the entropy falls to single digit numbers (the minimal value in the data is only 1 bit!).

Another observation is that the entropy falls to a very low level even during the installation process as opposed to the expectation of problems with entropy only at the beginning due to lack of the restored random data pool. Thus any simple solution like waiting for the available entropy to become high enough when the installer starts would not change anything, because the sensitive operations take place in later phases of the installation process.

As the gathered data and also the figure 3 show, the biggest problems with entropy (and thus potentially with (pseudo)random data quality) were detected on the laptop. That was the reason why most of the (pseudo)random data samples were gathered from that machine. Each of the 32MB samples was then analyzed with the **ent** tool [12] giving some basic statistics about the randomness and the samples were mutually compared to each other with the **bit-diff** [13] utility that has been created. It reads two files, given as command-line arguments, byte per byte and computes the number of bits in which the bytes (files) differ[7]. Two samples of truly random data are

[7]the so-called *Hamming distance* [14]

expected to differ in half of the bits[8]. The statistics[9] returned by the **ent** tool were quite surprisingly very good, all similar to the following example:

```
Entropy = 7.999995 bits per byte.

Optimum compression would reduce the size
of this 33554432 byte file by 0 percent.

Chi square distribution for 33554432 samples is 249.82,
and randomly would exceed this value 57.99 percent of the times.

Arithmetic mean value of data bytes is 127.4855 (127.5 = random).
Monte Carlo value for Pi is 3.141380497 (error 0.01 percent).
Serial correlation coefficient is 0.000032 (totally
uncorrelated = 0.0).
```

The results of the bit comparisons of the pseudorandom data samples from laptop are in the following table:

compared samples	**% of different bits**
#1 and #2	50.0043
#1 and #3	50.0025
#1 and #4	50.0042
#1 and #5	49.9981
#2 and #3	50.0002
#2 and #4	49.9952
#2 and #5	49.9990
#3 and #4	49.9992
#3 and #5	49.9980
#4 and #5	50.0013

These values show that all the samples mutually differed practically in a half of their bits and thus any correct cryptographic operation (with so-called *avalanche effect*) using them shouldn't have suffered from this data being too similar in multiple installations on such hardware. Because hardly ever some tool needs 32 MB of (pseudo)random data, we have also tried to compare the first 2 MB, 1 MB and 512 KB of the samples, but the results were basically the same as for the full samples.

[8]which is a simple mathematical exercise left to the reader
[9]explanations of the statistics can be found at [12]

4 Conclusions and patches

The very basic question that the evaluation of the gathered data implies is if there are really any problems with low entropy or low-quality of the (pseudo)random data possible or (probable) to happen during the installation process. On one hand, the statistics for the gathered data samples and their mutual comparison show that even on a system that starts with low amount of entropy in the kernel the pseudorandom data provided by the kernel has high quality as far as the check for similarity and basic statistics go. On the other hand, the entropy profiles (and the data behind them) show that some installations can really start with extremely low amounts of entropy in the kernel and even if the entropy level rises to better values, it quite often falls down during the installation process. That could potentially lead into problems in combination with the actions most sensitive to the quality of pseudorandom data they require.

Typical example of such action is a key generation. Be it generation of the keys for asymmetric cryptography (like SSH keys or keys for SSL certificates) or (master) keys for symmetric cryptography. While creation of many keys may be postponed to the post-installation phase were the system runs in a standard (non-installation) environment, disks encryption has to be set up in the installation process, because the installer needs to write data to such encrypted disks[10].

For this reason two sets of patches, one for the Anaconda installer and one for the *blivet* library used by the installer for storage manipulation and configuration, has been created. The purpose of these patches is to make sure that the available entropy in the kernel is high enough before a new *LUKS* (Linux Unified Key Setup) [15] device is created. That required adding a code to the *blivet* library that triggers registered callbacks in various places and a code for the Anaconda installer that provides a callback for the event of waiting for the entropy level to become high enough. As a video preview [16] of the new feature shows, a window informing user about what is going on and with a hint how they can help is raised and indicates the progress of current entropy level compared to the desired value. For the video preview 1500 bits of entropy were required as the minimal value to demonstrate the functionality of the feature, but setting the threshold to some lower, reasonable value will mean the screen will hardly ever appear and if yes, it will definitely not take such a lot of time to generate the required entropy.

[10]actually, it is possible to move data somewhere else after installation, encrypt the disk and move the data back, but it is a non-trivial and error-prone action which usually needs to be done in some special environment (where e.g. the main system disk or partition is not mounted and used)

The patches for the new feature were sent to the mailing list [17] [18] for a review and the intention is to push them to the stable repository so that they are included in the versions of the blivet library and the Anaconda installer used by the Fedora and RHEL GNU/Linux distributions. The best threshold value for the minimal entropy should be decided by the experts in this area and modified before the patched code starts to be used and an email conversation on that topic has been already started. The other sensitive operations that can potentially take place in the installation process are not done by the installer itself, but by scripts written by the users that will be instructed to include some "guard" for a minimal entropy level to their code. However if some of such operations will ever get integrated to the installer, it shouldn't be a problem to add another piece of code similar to the patches created as part of this work to prevent any potential issues with low entropy in the installation process.

All the gathered data as well as source codes of all the tools written as part of this work are available at author's web space [19].

References

[1] KRHOVJÁK, Jan. *Cryptographic random and pseudorandom data generators* [online] 2009, [cit. 2013-05-26]. Dissertation thesis. Masaryk university, Faculty of informatics. Advisor Václav Matyáš. Available at:

<`http://is.muni.cz/th/39510/fi_d/`>

[2] Stephan Mueller. *avoid entropy starvation due to stack protection* [online] 11.12.2012, [cited 24.05.2013]. Available at:

<`https://patchwork.kernel.org/patch/1861971/`>

[3] STEVANOVIĆ, Radomir. *Quantum Random Number Generator* [online] 2007, [cited 24.05.2013]. Available at:

<`http://qrbg.irb.hr/`>

[4] SCHNEIER, Bruce. *Schneier on Security: Lousy Random Numbers Cause Insecure Public Keys* [online] 16.02.2012, [cited 24.05.2013]. Available at:

<`http://www.schneier.com/blog/archives/2012/02/lousy_random_nu.html`>

[5] Fedora project community. *Anaconda wiki* [online] 16.05.2013, [cited 24.05.2013]. Available at:

<`https://fedoraproject.org/wiki/Anaconda`>

[6] Fedora project community. *Anaconda Updates wiki* [online] 06.07.2012, [cited 24.05.2013]. Available at:

<`https://fedoraproject.org/wiki/Anaconda/Updates`>

[7] Free Electrons. *Linux/include/uapi/linux/random.h – Linux Cross Reference* [online] , [cited 24.05.2013]. Available at:

<`http://lxr.free-electrons.com/source/include/uapi/linux/random.h#L14`>

[8] PODZIMEK, Vratislav. *entropy_profiler.c* [online] 08.04.2013, [cited 24.05.2013]. Available at:

<`http://www.fi.muni.cz/~xpodzim/git/?p=entropy-anaconda-addon.git;a=blob;f=entropy_profiler.c;h=cda180cbeb3250928ffcfb940a1869237c6c92e0;hb=HEAD`>

[9] PODZIMEK, Vratislav. *linux_rand.c* [online] 03.03.2013, [cited 24.05.2013]. Available at:

<`http://www.fi.muni.cz/~xpodzim/git/?p=entropy-anaconda-addon.git;a=blob;f=linux_rand.c;h=71a0e0011bc522ed14236018a0d55e066ecc8b54;hb=HEAD`>

[10] QEMU. *Features-Done/VirtIORNG* [online] 18.02.2013, [cited 24.05.2013]. Available at:

<`http://wiki.qemu.org/Features-Done/VirtIORNG`>

[11] SHAH, Amit. *About Random Numbers and Virtual Machines* [online] 11.01.2013, [cited 24.05.2013]. Available at:

<`http://log.amitshah.net/2013/01/about-random-numbers-and-virtual-machines/`>

[12] Calomel.org. *Entropy and Random Number Generators* [online] 13.03.2013, [cited 24.05.2013]. Available at:

<`https://calomel.org/entropy_random_number_generators.html`>

[13] PODZIMEK, Vratislav. *bit_diff.c* [online] 21.05.2013, [cited 24.05.2013]. Available at:

<`http://www.fi.muni.cz/~xpodzim/git/?p=entropy-anaconda-addon.git;a=blob;f=bit_diff.c;h=80d3314a4aa4708b4fa211d2db817f3141eb712a;hb=HEAD`>

[14] Wikipedia contributors. *Hamming distance* [online] 13.05.2013, [cited 24.05.2013]. Available at:

<http://en.wikipedia.org/wiki/Hamming_distance>

[15] Wikipedia contributors. *Linux Unified Key Setup* [online] 27.02.2013, [cited 24.05.2013]. Available at:

<http://en.wikipedia.org/wiki/LUKS>

[16] PODZIMEK, Vratislav. *ensure_ entropy.webm* [online] 24.05.2013, [cited 24.05.2013]. Available at:

<http://vpodzime.fedorapeople.org/ensure_entropy.webm>

[17] PODZIMEK, Vratislav. *[blivet] My school project – random data entropy in installation* [online] 24.05.2013, [cited 24.05.2013]. Available at:

<https://lists.fedorahosted.org/pipermail/anaconda-patches/2013-May/004281.html>

[18] PODZIMEK, Vratislav. *[master] School project – random data entropy* [online] 24.05.2013, [cited 24.05.2013]. Available at:

<https://lists.fedorahosted.org/pipermail/anaconda-patches/2013-May/004282.html>

[19] PODZIMEK, Vratislav. *Resources for the random data entropy in the OS installation* [online] 25.05.2013, [cited 25.05.2013]. Available at:

<https://www.fi.muni.cz/~xpodzim/PA018/entropy_in_OS_installation/>